I hereby certify that this paper is being deposited with the United States Postal Service as Express Mail in an envelope addressed to: Box Patent Application, U.S. Patent and Trademark Office, P.O. Box 2327, Arlington, VA 22202, on this date.

November 28, 2001 [illegible]
Date *Express Mail Label No.: EL 846163174 US*

APPLICATION FOR LETTERS PATENT OF THE UNITED STATES

Mark Gagner
30W515 Diversey Parkway
West Chicago, IL 60185

APPARATUS AND METHOD FOR EXECUTING BLOCK PROGRAMS

TO WHOM IT MAY CONCERN, THE FOLLOWING IS A SPECIFICATION OF THE AFORESAID INVENTION

APPARATUS AND METHOD FOR EXECUTING BLOCK PROGRAMS

BACKGROUND

The present invention generally relates to block programming, and more particularly to block programming in which the blocks in a program are selectively executed based on changes in the input values of the blocks.

Block programming is well known in the control industry. In contrast to the traditional control programs using sequential language to write lines of codes, block programming is done generally through drawings. For those who are in the control industry, such as building managers or building operations system dealers who install control systems, it is generally more intuitive and easier to draw the control strategies pictorially into blocks that perform various functions, rather than writing it in some programming language such as C or BASIC, for example. These blocks are compiled into a file of records that describe the operations or functions of the blocks, and an interpreter or execution engine interprets the records. Generally, each block in the program represents a subroutine or algorithm that performs various specified tasks. These blocks are connected together through input and output lines or connectors to form a complete program.

Typically, the control systems that employ block programs execute all the blocks within the program each time the program is run. In a block program for operating a heating/ventilation/air conditioning (HVAC) system, for example, there could be from about 1,000 to 2,000 blocks, each of which must be executed every run of the program, which could be every few hundred milliseconds to give the appearance of smooth real-time operation to a human observer. Performing these executions in the required time is not a problem if a relatively fast computer is used. However, such computers are expensive and add to the cost of the control system.

SUMMARY OF THE INVENTION

The present invention is directed to a block programming environment for selectively executing a series of function blocks that make up a program. During each cycle of the program's operation, a determination is made as to whether any input value of each function block has changed, and only those blocks having a changed input are executed. In this manner, a significant economy in programming processing time is realized, thereby allowing use of less powerful and inexpensive processors.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a block diagram of a block program execution system in accordance with the present invention;

FIG. 2 is an example of a pictorial representation of a mathematical operation block in a block program;

FIG. 3 is a block diagram showing the connection between external input and output blocks and the I/O subsystem shown in FIG. 1;

FIG. 4 is a format of a block table shown in FIG. 1;

FIG. 5 is a format of a connector table shown in FIG. 1;

FIG. 6 is a flowchart illustrating the operation of a block execution engine of FIG. 1;

FIG. 7 is a flowchart illustrating the process for executing external inputs by the block execution engine;

FIG. 8 is a flowchart illustrating the process for executing the blocks; and,

FIG. 9 is a flowchart illustrating how changes are propagated through connectors in the block program.

DETAILED DESCRIPTION

Broadly stated, the present invention is directed to an apparatus for executing a block program, and includes a block table listing records corresponding to the blocks in the block program. A block library holds algorithms associated with the blocks, and an executing program process the blocks in the block program in accordance with the associated algorithms. In running the block program, the executing program selectively executes the blocks that receive a new input value which is different from a previous input value.

In accordance with another aspect of the present invention, a method for executing a block program includes creating a table of block records that correspond to a plurality of blocks used in the block program, and creating a library for holding algorithms for executing functions associated with the blocks. The method further includes selectively setting a flag in the block records when at least one input value of the corresponding blocks changes, and then executing the algorithms of the blocks having corresponding block records that have the flag set.

Turning now to FIG. 1, a block program execution system in accordance with the present invention is shown generally at 10, and includes a block execution engine 12 that is operatively connected to an execution image file (EIF) 14, a block table 16, a connector table 18, at least one input/output (I/O) subsystem 20 and a block library 22. The block program execution system 10 is adapted to selectively execute a series of blocks that perform various functions and are arranged and connected in a particular order to make up a block program. The blocks are essentially algorithms for performing various functions, for example, additions, subtractions, PID control, etc. As such, each block includes at least one input and output, as shown in FIG. 2 of a block 24 that performs mathematical additions.

In the preferred embodiment, the block program execution system 10 is implemented in a programmable equipment controller 23 in a control network for controlling, among other things, various end devices such as, for example, fans or actuators in a heating/ventilation/air conditioning (HVAC) system. It should be understood, however, that the present block program execution system 10 may be employed in various other controller devices.

The block execution engine 12 is a firmware/software program written preferably in C++ object oriented programming language, and is responsible for initializing and executing a block program associated with the block program execution system 10. At the start-up of the system 10, the block execution engine 12 reads the EIF 14, which contains descriptions of the blocks and connectors for operatively connecting the blocks within the block program, and constructs the block table 16 and the connector table 18. As shown in FIG. 3, the I/O subsystem 20 is used to input values into the execution engine 12 from external input blocks 25 (one shown) and to output values to external output blocks 27 (one shown) from the execution engine. In a HVAC system, for example, the external input block 25 might represent a temperature reading from a certain location, and the external output block 27 might represent a signal supplied to an actuator for opening and closing a valve in a hot or cold water pipe. The block library 22 includes a collection of subroutines or algorithms needed to execute the function blocks. For example, the "Add" block 24 shown in FIG. 2 has a corresponding algorithm in the block library 22 for calculating the sum of two input values.

Turning now to FIG. 4, the block table 16 includes a series of records 26, one record for each block in the block program. The records 26 are sorted in the order of execution of their corresponding blocks in the block program. It should be noted that in the preferred embodiment, the records 26 are already listed in their proper execution order in the EIF 14 even before the block table 16 is constructed.

Each record 26 in the block table 16 provides a block type field 28 for indicating the algorithms that the corresponding block performs, for example, additions, subtractions, PID control, etc. A block ID field 30 provides the block identification number for the block. Input connector ID fields 32 (two shown in FIG. 4) identify each input connector that supplies input values to the block, and output connector ID fields 34 identify the output connectors for carrying output values of the block. Input value fields 36 (two shown) hold values that are input to the block, and output value fields 38 (one shown) hold the output values. It should be noted that the number of input value fields 36 corresponds to the number of input connector ID fields 32, and the output value fields 38 to that of the output connector ID fields 34. Only one value is stored in each of the input and output value fields 36, 38. It should also be noted the external input blocks 25 do not have input connectors because the values for these blocks are provided via the I/O subsystem 20. Similarly, external output blocks 27 do not have output connectors because the value is sent to the I/O subsystem 20. Accordingly, the records 26 for the blocks that are operatively connected to the external input or output blocks 25, 27 do have input and output connector ID fields 32, 34 or the input and output value fields 36, 38.

Each record 26 also includes a field 40 ("BlockExecEnabled) for setting a flag indicating whether the block should be executed. This field 40 is set to TRUE to indicate that the block should be executed when at least one of the input values 36 of the block experiences a change, and to FALSE to indicate that the block should not be executed, when no input value of the block has changed.

Turning now to FIG. 5, the connector table 18 also includes a series of records 42, one each for every connector 44 (best shown in FIG. 2) in the block program for operatively connecting the blocks together. It should be noted that only one connector is connected to a block output and may be connected to one or more inputs of subsequent or destination blocks. Each connector record 42 includes a connector ID field 46 providing an identification number for the corresponding connector in the program. A source block field 48 provides the block identification of the block from which the connector originates, and destination block ID fields 50 identify one or more blocks at which the connector terminates.

Turning now to FIG. 6, once the block table 16 and the connector table 18 have been established, the block execution engine 12 waits in a suspended state until the start of a new period (step 52), then processes external inputs from the I/O subsystem 20 (step 54), and executes the blocks in the program that are affected by the external inputs (step 56).

More specifically and turning to FIG. 7, the execution engine 12 processes the external inputs from the I/O subsystem 20 by determining whether an I/O message is available in the queue at the start of a new interval (step 58). The block execution engine 12 receives and queues a message from the I/O subsystems 20 if the external input block 25 has experienced a change of input value. If no message is queued, the execution engine 12 waits until the next execution interval to process the messages, if any (step 60). If a message is available, the execution engine 12 fetches the message from the queue (step 62), examines the message to determine which block in the program has experienced a change in its input value (step 64), and sets a flag in the BlockExecEnabled field 40 in the record 26 in the block table 16 to TRUE for the corresponding block (step 66). This sequence is repeated for each message in the queue.

Turning now to Fig. 8, the manner in which the execution engine 12 executes the blocks in the block program is described. The execution engine 12 goes to the first block listed in the block table 16 (step 68), and determines whether the BlockExecEnabled field 40 has been set to TRUE, indicating that at least one input value to the block has experienced a change (step 70). If not, the execution engine 12 advances to the next block 24 on the list (step 72). Then it is determined whether all the blocks 24 have been checked (step 74). The process ends at this point if all the blocks have been checked (step 76). If, however, not all blocks 24 have been checked, then the process goes back up to step 70.

If at step 70, it is determined that the BlockExecEnabled field 40 has been set to TRUE, the execution engine 12 obtains the corresponding algorithm for the block from the block library 22 and executes the algorithm (step 71). After the block has been executed, the BlockExecEnabled field 40 of that block is set to FALSE (step 73). The execution engine 12 then goes to the first output of the executed block (step 75), and determines if all the outputs have been checked to ascertain whether the value of that output has changed as a result of running the algorithm for that block (step 78). If so, the process advances to the next block (step 72).

If all the block outputs have not been checked, then the execution engine 12 determines whether the value of that output has changed (step 80). If not, the execution engine 12 advances to the next output of the same block (step 82), where a determination is again made as to whether all the outputs of that block have been checked (step 78).

Referring to Fig. 9, if the value of the output has changed at step 80 (shown in Fig. 8), the execution engine 12 goes to the first destination block listed in the record 42 in the connector table 18 corresponding to the executed block (step 84). The execution engine 12 then sets a TRUE flag in the BlockExecEnabled field 40 of the record 26 in the block table 16 corresponding to the first destination block (step 86). Following this operation, it is determined if the current destination block is the last block listed in the connector record 42 (step 88). If not, the execution engine 12 goes to the next destination block listed on the connector record 42 (step 90), where a TRUE flag is set in the BlockExecEnabled field 40 for that destination block (step 86). If at step 88, the previous block in which a TRUE flag was set is the last block, then the process returns to block 82 (see FIG. 8), where the execution engine 12 advances to the next output of the executed block (step 82).

From the foregoing description, it should be understood that an improved system for executing a block program has been shown and described which has many desirable attributes and advantages. The block execution engine goes through each block in the block program in the order that the blocks are arranged and determines whether any input value has changed. In accordance with the present invention, only those blocks that have experienced a change in their respective input values are executed, thereby increasing operating efficiency.

While various embodiments of the present invention have been shown and described, it should be understood that other modifications, substitutions and alternatives are apparent to one of ordinary skill in the art. Such modifications, substitutions and alternatives can be made without departing from the spirit and scope of the invention, which should be determined from the appended claims.

Various features of the invention are set forth in the appended claims.